Team Controller
Northern Arizona University
Flagstaff, Arizona
October 20th, 2023

Zachary Parham (Team Lead): zjp29@nau.edu
Italo Santos (Mentor): ids37@nau.edu
Bradley Essegian: bbe24@nau.edu
Brandon Udall: bcu8@nau.edu
Dylan Motz: djm658@nau.edu

Requirements Specification

Northrop Grumman

Weapon System Support Software

Harlan Mitchell

Laurel Enstrom

Accepted as baseline requirements for this project by:

Client :	____________	____________
Team Lead :	____________	____________
	Date	Signature

Table Of Contents

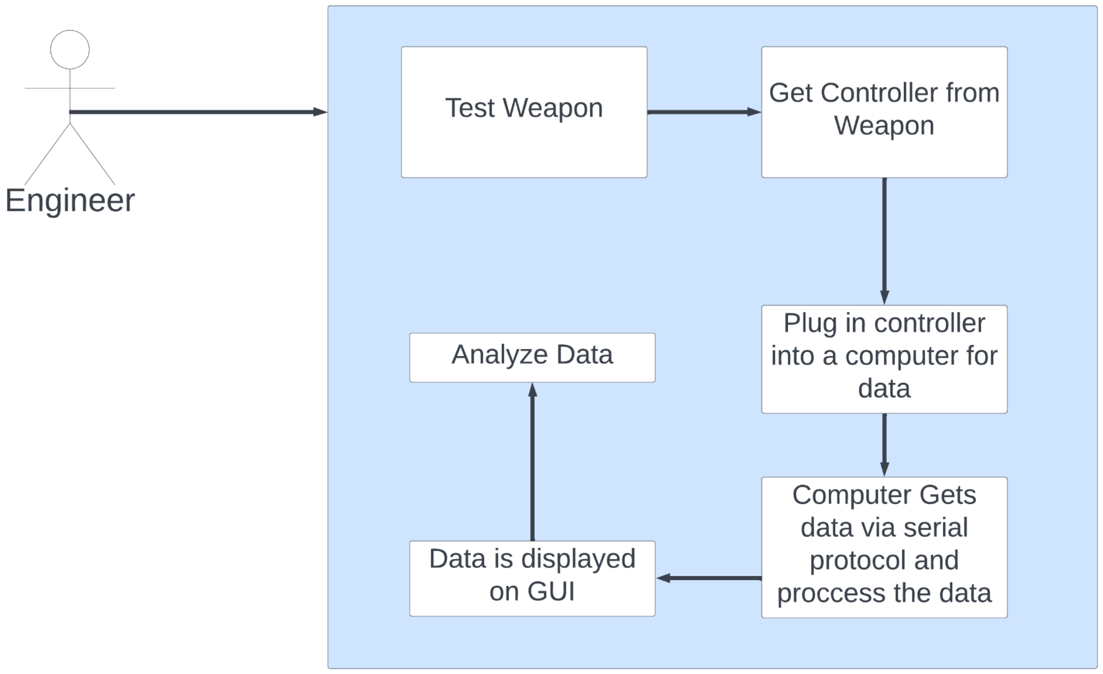
1.0 Introduction

In the realm of aerospace and defense technology; where precision, security, and reliability are of utmost importance, the need to effectively diagnose and rectify issues with military weaponry is paramount. Multi-billion dollar defense contractors are equipping the United States with products ranging from advanced military aircraft to state-of-the-art weaponry. This defense sector plays a vital role in national security, global stability, and is substantial in terms of economic impact. Every year these systems become increasingly more complex and must stay reliable.

Northrop Grumman, our clients, stand as a prominent figure in the aerospace and defense technology industry. A cornerstone in the American military, they have built everything from the B-2 Stealth Bomber to the James Webb Space Telescope [1]. It's no surprise that these revolutionary projects create an endless amount of diagnostic data. With their formidable presence, they provide a spectrum of armament systems and services to their customers in over 25 nations around the world [2]. Northrop Grumman's success is not solely measured in monetary terms; they play a critical role in the national security and well-being of military personnel worldwide. The heart of the challenge that our clients face lies in the diagnostics of armament systems. Sr. Systems Engineer Manager, Harlan Mitchell, and Principal Systems Engineer, Laurel Enstrom, are faced with a cumbersome and resource-intensive process for obtaining diagnostics on their products when the end-user has a problem. This document will outline the requirements for our software that shall transform this process, enabling maintenance personnel and their customers to easily obtain the best diagnostic data possible regarding the weapon system. In today's climate, ensuring that our military armament systems are functioning optimally is not just a matter of national security; it is a global concern.

2.0 Problem Statement

Our clients need to test weapons that are developed by many different engineers and to do this they utilize a controller that gets the weapons data and a software product to process this data. The clients have many issues with this software product since it's complicated. The process of how the software product they currently have works like this.


Engineer
Test Weapon
Get Controller from Weapon
Plug in controller into a computer for data
Computer Gets data via serial protocol and proccess the data
Data is displayed on GUI
Analyze Data

At Northrop the current issue with the software product they currently have is that it is complicated so that only engineers can understand the program and the people in Yuma don't know how to use the software. Also sending a team of engineers down to Yuma to test a weapon every time takes away time and resources from other engineering tasks. Finally it also needs admin permissions to install the software.

The current inefficiencies and missing elements in the client's current software product:

- Very complicated GUI that only engineers can understand
- No way for Yuma workers to use/communicate the data back to the engineers in Phoenix
- Needs admin permission to install

3.0 Our Solution: A Graphical Data Display

To help our clients diagnose problems within their weapon systems, we propose a desktop application which can read serial data sent from a weapon controller and display it in a user friendly way. The weapon controller is already capable of generating and sending data such as firing mode, firing rate, weapon position (in degrees) etc in the form of RS422 serial messages. Our application must be able to read this data and use it to generate a comprehensive graphical report which will keep weapon operators, engineers and technicians informed on the current status of their system.

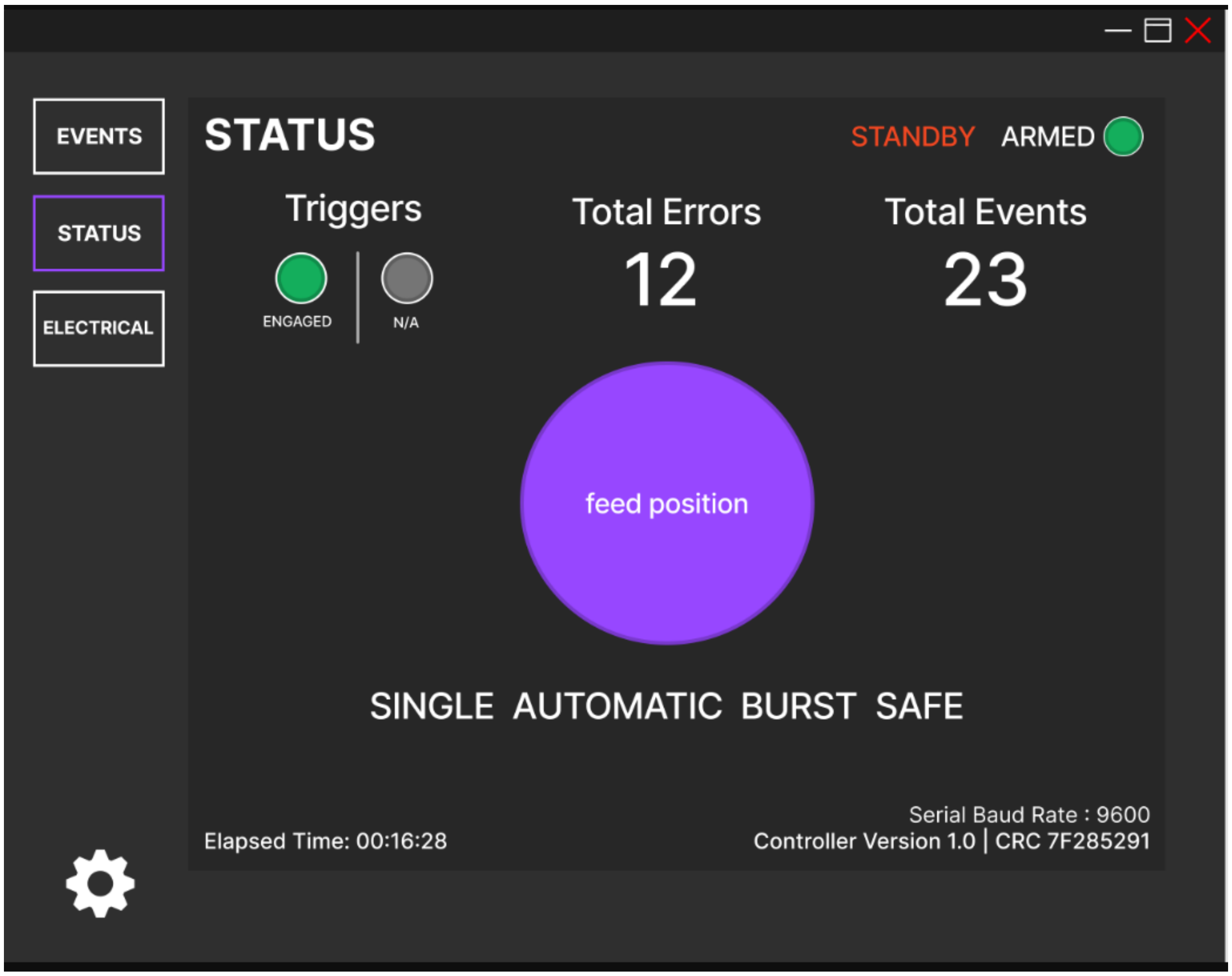
Our application will display the following data:

- Weapon status (armed or disarmed)
- Serial baud rate
- Record of errors encountered since weapon activation (overheat, jam, mechanical failure, etc.)
- Record of events logged since weapon activation (firing events, reload events, etc.)
- Time elapsed since weapon activation
- Current Firing mode (automatic, burst, single, safe)
- Current burst length (number of concurrent launches per firing event)
- Current firing rate (number of launches per second weapon is capable of)
- Estimated feed position (weapons current state in firing cycle i.e. ejecting spent shell, feeding new shell, firing etc.)
- Electrical components (a list of various components including voltages and currents for each)

Some categories of data such as feed position and trigger status can be represented graphically. Other data categories such as records of errors and events will need to be logged in text format. Our goal is to find the most intuitive and simple way of representing each data category.

We must also streamline user navigation between different menus and data displays. This will make our application easy to learn and allow users to quickly gain access to any piece of information we track.

Below is a sample rough draft design of the status section of our GUI. This graphic is subject to change as development of this project progresses.



Elapsed Time: 00:16:28

Serial Baud Rate : 9600
Controller Version 1.0 | CRC 7F285291

Additionally, our project must implement cybersecurity concepts to eliminate the possibility of misuse. This means reducing the level of user interactions with our system to very predictable and well guarded inputs most often in the form of buttons and dropdown menus.

Our envisioned software product will lead to more effective weapon operation and understanding whether that be in the field or during testing/repair.

4.0 Project Requirements

In this section we will introduce specific requirements that our finished product must satisfy. These requirements must be verifiable or in other words we must be able to measure some aspect of our product to undeniably prove that they are satisfied.

The names of each requirement ex. CR01 (client requirement 01) and SR01 (software requirement 01) represent their scope. To put it simply, client requirements are the requirements we derived directly from the needs of our clients. They represent things the client cares about. Updates given to our clients throughout the course of development will reference these requirements. Software requirements are requirements derived from taking a closer look into what kind of software tools will be needed to complete our project. Since clients are unlikely to care about the low level specifics, these requirements will often be omitted during discussions with the clients.

Some terminology used within this section may be unknown to readers. View the Glossary in section 8.0 for definitions of technical terms.

4.1 Client Level Requirements

Client level requirements represent the specific items our clients want from our finished product. These requirements along with the environmental requirements outlined in section 4.2 give us the grounds for developing the functional requirements.

- **CR01 -** The data display module shall be a desktop application.
- **CR02 -** The data display module shall read input data via RS422 serial protocol from the controller simulator.
- **CR03 -** The data display module shall have the ability to write the following data into a log file…
    a. All event strings recorded up to the point in which the user requests the log file. (this is the case for generating a log file during a session)
    b. All event strings recorded throughout a complete session. Up to 5 complete session log files are kept before overwrites occur. (this is the case for automatically generating a log file after a session)
- **CR04 -** The data display module shall display all status information directly to the application's window for the duration of a session.
- **CR05** - Requests for status updates will be sent by data display module to the controller simulator every 0.25 seconds.
- **CR06 -** The data display module shall not require admin rights to install, set up, or use.
- **CR07 -** The data display module shall include filtering options to filter events and errors along the following categories:
    a. show only errors
    b. show only cleared errors
    c. show only active errors
    d. show only events

Since our client level requirements are self documenting and have been briefly demonstrated in the solution section, there is not much need for further explanation. These requirements simply exist as a precise mode of defining what our conditions of success are in this project. Additionally these requirements allow us to foster a concrete understanding with our clients as to what Team Controller's obligations are so that we may better provide the services they have requested.

4.2 Environmental Requirements

Environment requirements represent constraints imposed by the environment in which the clients want our finished product to be deployed in. For this project our environmental requirements are as follows:

- **CR07 -** The system and its environment shall be installed via an installer file.
- **CR08 -** The system shall be portable on devices running windows 10 or 11.
- **CR09G -** The system should be portable on Debian Linux distributions.
- **CR10 -** The system shall include a controller simulator which will generate and send status updates and events to the data display module for testing purposes.

CR07:

Many of the features of our project require assets such as images, supporting files and custom directories. We will need to install these assets into the host system in order for the project to operate as intended. That is where CR07 becomes a necessity. Using an installer will allow us to generate a proper environment for our system to be deployed in. We plan to create an assets directory and a log directory. Additionally a main project directory will be generated to store all installed data including our data display module and the previously mentioned directories. While running, our data display module can navigate these directories and the files they hold to perform its operations. If the user would like to uninstall the application all directories and data must be deleted using the same installer file that was used to deploy the project.

CR08:

The client intends to run our product on computers using mainly Windows 10 and 11. We must ensure that all of the technology we use for development is compatible with these platforms.

CR09G:

Our client also informed us that some of their computer systems operate in linux environments. As a stretch goal we have decided to consider technologies that are cross compatible with windows and linux to give our clients a greater degree of flexibility when working with our product. Considering REQ09G is a stretch goal, it is not required to be satisfied by our minimum viable product.

CR10:

While our client currently has software capable of feeding weapon data to our data display module, we do not have access to this software for purposes of confidentiality. Therefore in order to test our data display module, we must develop a controller simulator which will emulate the controller software currently in use by our client. After our project has been completed, our controller simulator along with our data display module will be handed over to Northrop Grumman for further development of the system. As it stands, the controller simulator we develop will only need to include capabilities for generating status and event data then delivering it to our data display module through the use of RS422 serial communication.

4.3 Performance Requirements

Performance requirements represent specific quotas which relate to the functionality discussed in previously defined requirements. These are things like the refresh rate of our GUI, and serial communication speed. Specific numbers relating to performance were provided in the client level requirements in section 4.1 and will also be included in our functional requirements in section 4.4.

- **PR01 -** The application shall be designed with object oriented programming principles to ensure readability of the application.
- **PR02 -** The application shall be designed with object oriented programming principles to ensure scalability of the application.
- **PR03 -** The application shall be designed with object oriented programming principles to ensure modularity of the application.

PR01:

Once development of the application is complete, the team will hand over the source code to developers at Northrop Grumman. To ease their development process, the application should be readable. This will include: class names, variable names, and method names. The team will include comments as needed.

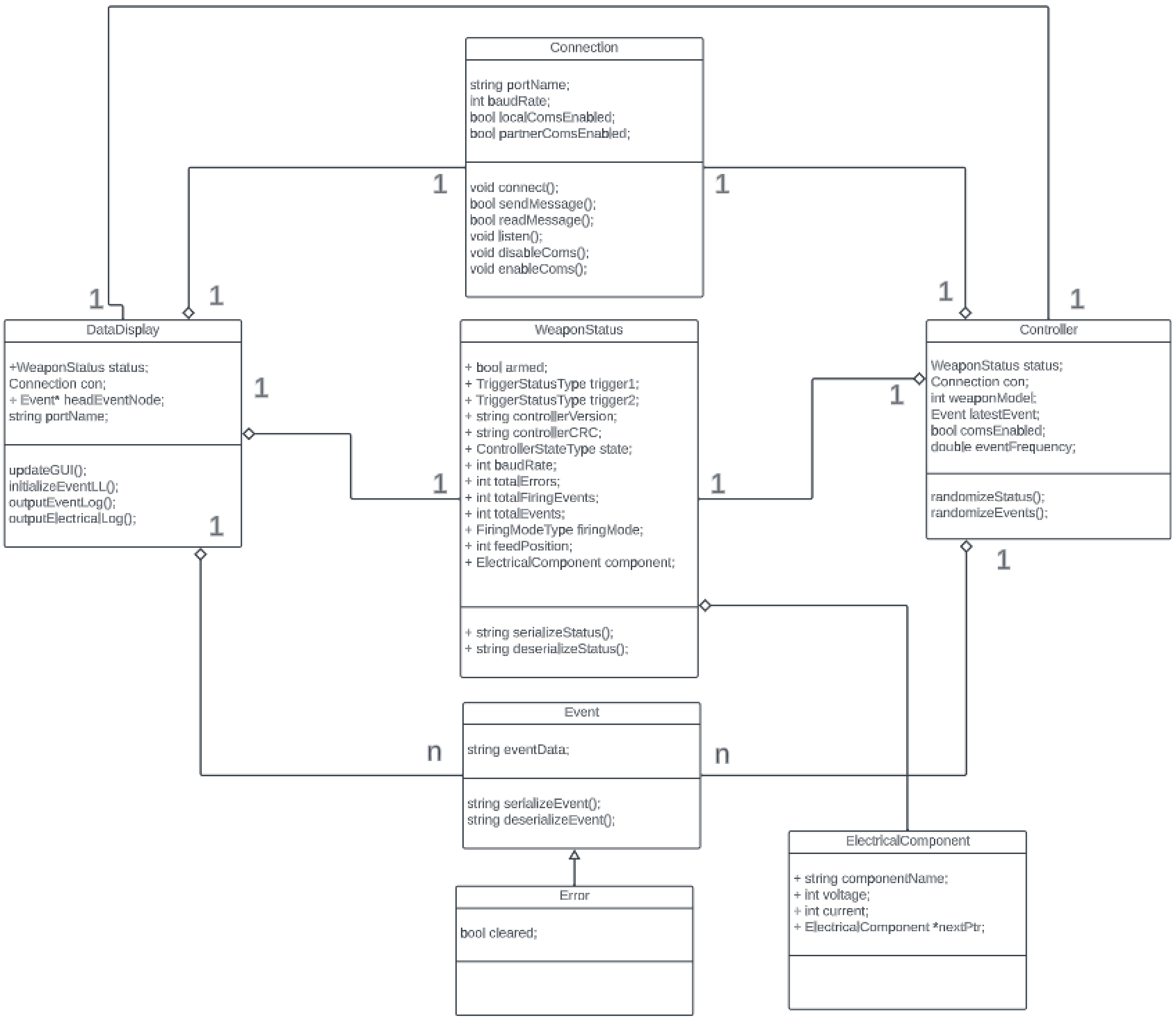
PR02:

Like **PR03**, the team intends on designing the application in such a way that ensures for easy scalability. Using object oriented programming, the team will design the application in a modular way.

PR03:

4.4 Functional Requirements

Functional requirements represent software functions we will develop as a means to satisfy all of the previously defined requirements. This section will be divided into specific subsystems that all contribute to the overall project. These subsystems are closely related to one another and there is often overlap between the functions needed for each. To give you an idea of how these systems work with one another, a class diagram will be provided along with an explanation included within each section.



4.4.2 Serial Communication

In order for our project to be successful, we must develop a few simple serial utility functions that will support the communication between the controller simulator and the data display module.

- **SR01 -** The functions serializeStatus() and serializeEvent() shall be capable of generating bit string versions of given status data and event data respectably.
- **SR02 -** The functions deserializeStatus() and deserializeEvent() shall be capable of generating the original status data and event data, given a serialized bit string.
- **SR03 -** The connect() function shall be capable of performing handshake protocols with another device running connect().
- **SR04 -** The function sendMessage() shall be able to transmit a serialized string through the serial port.
- **SR05 -** The function readMessage() shall be able to detect incoming serialized messages from the serial port and update the status class with new data.
- **SR06** - The function disableComs() shall be able to pause serial communication.
- **SR07 -** The function enableComs() shall be able to resume serial communication.

SR01:

When working with serial communication it is important to realize that computers use bit strings (strings of 1s and 0s) to store data. To be able to send any kind of data through a serial port, we must develop a method of translating data like integers and strings into bit strings.

SR02:

After we have received a serialized message, we must also be able to restore it to its original format so that it may be used within the program.

SR03:

Handshake protocols, defined in the glossary section, are essential for making sure communicating programs are on the same page. They are used to synchronize communication to prevent themselves from interrupting one another or misinterpreting data.

SR04:

This function will be called after a serialized string has been developed, it will ensure that the message is transmitted to the other program and ensure that the other program understands the message.

SR05:

ReadMessage is the concurrent process responsible for listening on the serial port and recording data received. This function will work with sendMessage to facilitate proper transportation of bitstring from one device to another.

SR06 & SR07:

The weapon controller is an important piece of equipment. It is responsible for running the weapon system, and sometimes it will be overwhelmed with this task. In this case it is ideal that communication between the controller and data display module should be stopped until the controller is less busy. These functions give the controller a way to tell the data display module that it is busy and unable to send data until further notice, and re-enable communication when it is ready to do so. Adding this functionality to our controller simulator and data display module is important so that our data display module is better prepared to work with real weapon controllers when the time comes.

4.4.2 Data Display Module

The data display module is the brains behind the GUI. It is responsible for updating data fed into the GUI and communicating with the controller simulator to get the updated data. It must also process user inputs into the GUI and perform file output operations. Additionally it must be error resistant and possess multithreading capabilities to do all of these things at once. Considering the data display module relies heavily on the use of the serial communication functions, GUI functions and object oriented programming, the list of functional requirements exclusive to only the data display module will be quite limited in size.

- **SR08 -** The function initializeEventLL() shall be capable of creating a head node for a linked list given an event string.
- **SR09** - The function initializeElectricalLL() shall be capable of creating a head node for a linked list given a set of electrical data. Then adding an unknown number of new nodes for subsequent electrical components.

- **SR10** - The function addEvent() shall be capable of adding a new event to the event linked list.
- **SR11 -** The function outputEventLog() shall be capable of writing a log file, given a linked list of event data.
- **SR12 -** The function outputElectricalLog() shall be capable of writing a log file, given a linked list of electrical component data.

4.4.3 Controller Simulator

As mentioned previously the controller simulator is responsible for creating and sending the data that will be used by our data display module which will be running the GUI. For complete testing capabilities, the controller simulator will need to be able to take in requests for unique data to be generated. For mass testing capabilities, the controller simulator will need to be able to generate random data continuously using random rates of generation.

- **SR13 -** The function randomizeEvents() shall be capable of generating random event strings out of a given set of events at random time intervals within a given range.
- **SR14 -** The function randomizeStatus() shall be capable of generating random status updates every 0.1 seconds.
- **SR15 -** The function customEvent() shall be capable of taking in user input data and generating an event string using this data.
- **SR16 -** The function customStatus() shall be capable of taking in user input data and updating the status data structure using this data.
- **SR17 -** The function getInput() shall be capable of listening for user input data and deciding what to do with it.

4.4.1 Graphical User Interface

Dylan will do this section once more research is done, and we know what framework we are using

Buttons functions (function for each button)

- **SR01 -**
- **SR02 -**

5.0 Potential Risks

Dylan

6.0 Project Plan

Gant chart and explain

7.0 Conclusion

Zach

8.0 Glossary

Brandon

Define ambiguous terms used throughout the document here